The Decompilation Problem

Catz Exchange 2026

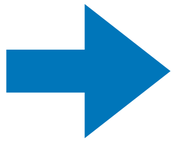
Louis-Emile Ploix — 26/02/26

Compilers

Compilers are software that make programs written by humans understandable to a computer.

Human

```c
int main() {
  printf("Hello World!\n");
}
```



Computer

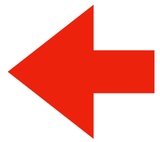
```
main:
  push    rbp
  mov     rbp,rsp
  lea     rdi,[str]
  call    printf
  mov     eax,0x0
  pop     rbp
  ret
```

Decompilers

Decompilers are software that takes code that computers understand, and turns it back into programs understandable to humans.

Human

```
int main() {
  printf("Hello World!\n");
}
```



Computer

```
main:
  push    rbp
  mov     rbp,rsp
  lea     rdi,[str]
  call    printf
  mov     eax,0x0
  pop     rbp
  ret
```

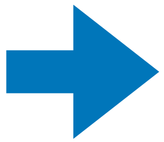
Why decompile?

- **Security research:** Stopping viruses, and finding and fixing vulnerabilities in closed source software.
- Security competitions (really!) and education.
- **For evil:**
  - Writing exploits against closed source software
  - Piracy / Game cheating / etc
- Because it exists anyway, so we may as well do it in the open.

Why is it difficult?

- Compilation loses a lot of information
  - Names of variables
  - **Structure of code and data**
  - Pretty much everything else…

```
int calculate_price(
  int apples, int oranges, int bananas
) {
  return
     apples * price_of_apples +
     oranges * price_of_oranges +
     bananas * price_of_bananas;
}
```



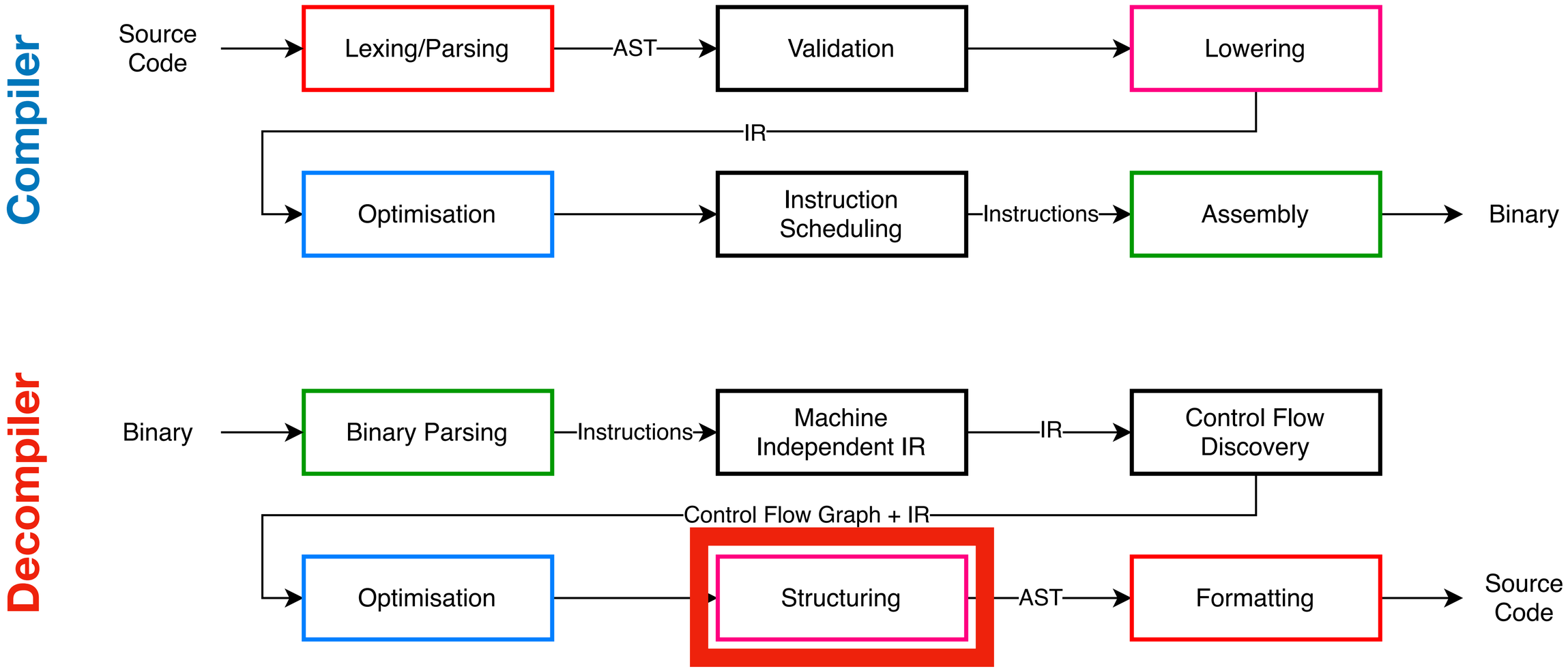
```
calculate_price:
  push   rbp
  mov    rbp,rsp
  mov    DWORD PTR [rbp-0x4],edi
  mov    DWORD PTR [rbp-0x8],esi
  mov    DWORD PTR [rbp-0xc],edx
  mov    eax,DWORD PTR [rip+0x0]
  imul   eax,DWORD PTR [rbp-0x4]
  mov    edx,eax
  mov    eax,DWORD PTR [rip+0x0]
  imul   eax,DWORD PTR [rbp-0x8]
  add    edx,eax
  mov    eax,DWORD PTR [rip+0x0]
  imul   eax,DWORD PTR [rbp-0xc]
  add    eax,edx
  pop    rbp
  ret
```

In fact, it’s not even possible!

- In general, most advanced program analysis is actually **undecidable**
- This is a concept from automata theory, which means that some problems cannot be solved in general by **any** algorithm
- But we can usually do *well enough* to be useful

So how do we do it anyway?

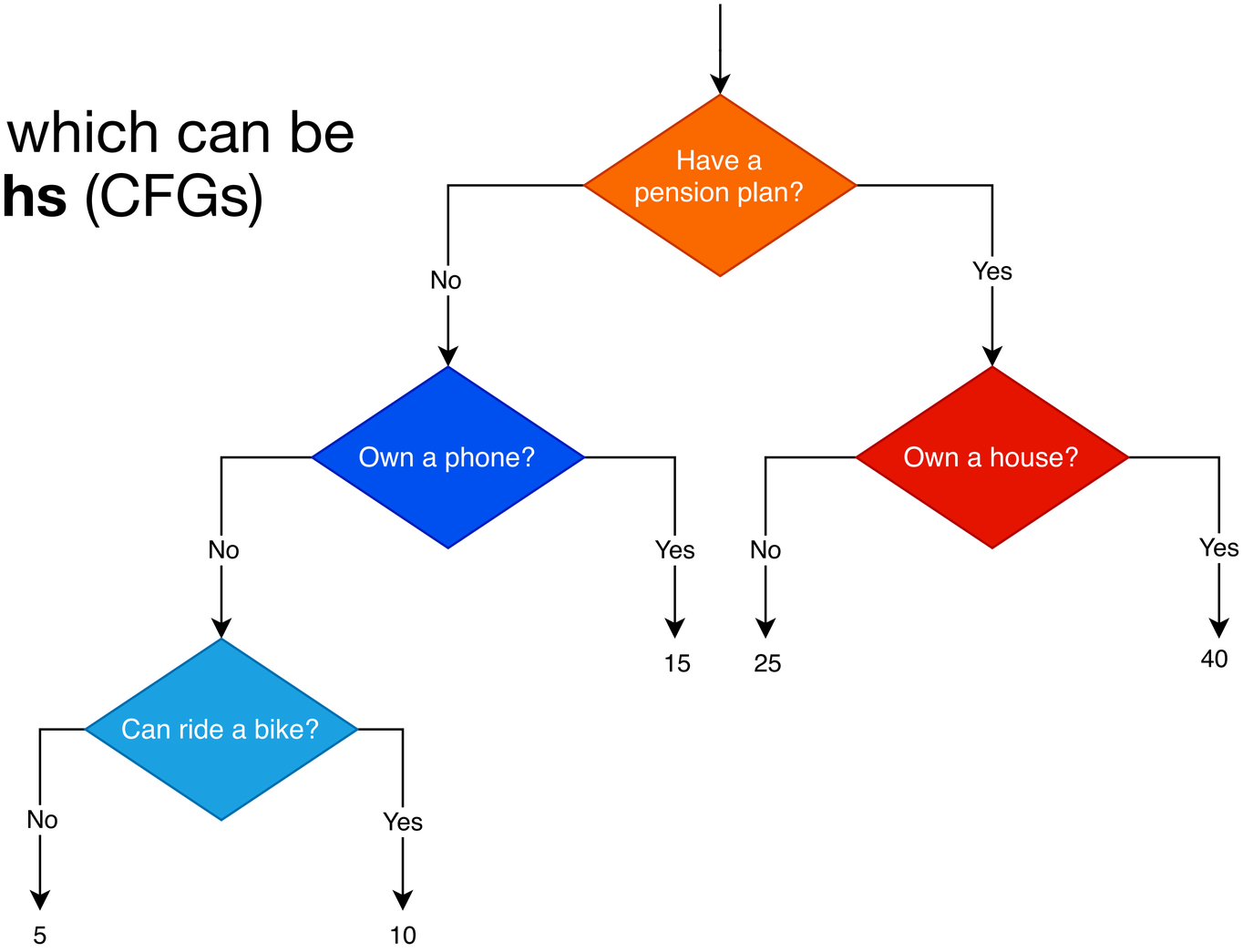
Roughly speaking, we just go backwards… Which is kind of the same as going forwards?



Control Flow

- Programs need to make **choices**, which can be represented as **control-flow graphs** (CFGs)

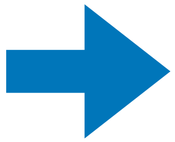
```
int guess_your_age() {
  if (has_pension)
    if (owns_house) return 40;
    else return 25;
  else
    if (owns_phone) return 15;
    else
      if (can_ride_a_bike) return 10;
      else return 5;
}
```



Control Flow

- A compiler takes away the structure of those choices, **flattening** the CFG

```
int guess_your_age() {
  if (has_pension)
    if (owns_house) return 40;
    else return 25;
  else
    if (owns_phone) return 15;
    else
      if (can_ride_a_bike) return 10;
      else return 5;
}
```

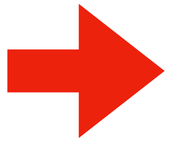


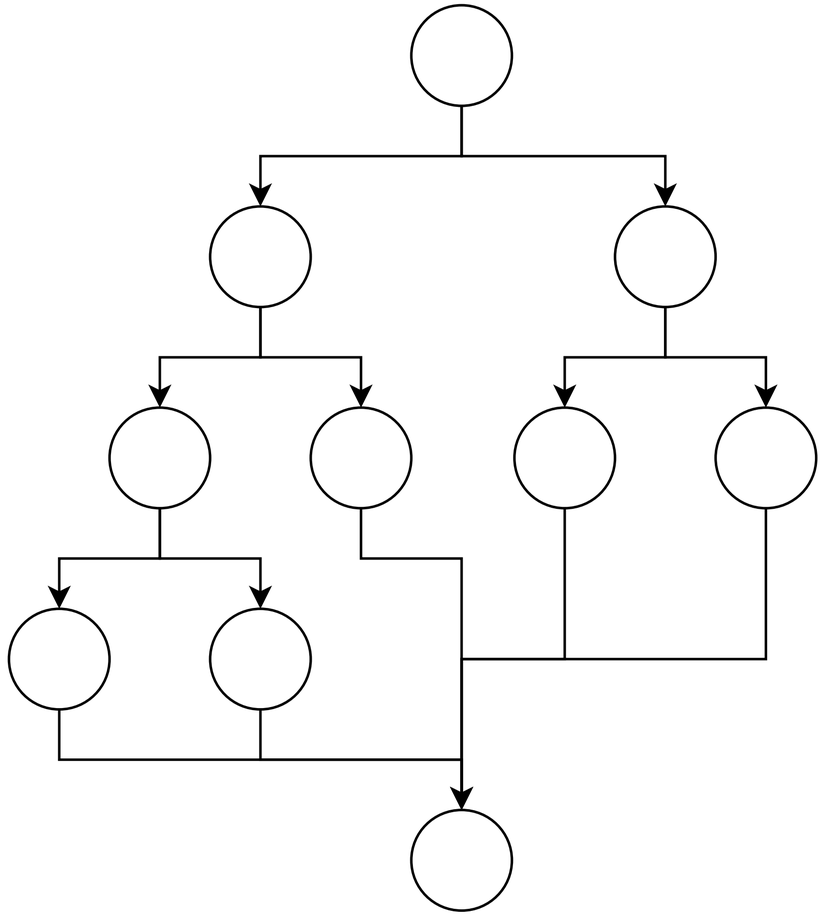
```
guess_your_age:
  push   rbp
  mov    rbp,rsp
  mov    eax,DWORD PTR [rip+0x0]
  test   eax,eax
  je     <guess_your_age+0x26>
  mov    eax,DWORD PTR [rip+0x0]
  test   eax,eax
  je     <guess_your_age+0x1f>
  mov    eax,0x28
  jmp    <guess_your_age+0x4d>
  mov    eax,0x19
  jmp    <guess_your_age+0x4d>
  mov    eax,DWORD PTR [rip+0x0]
  test   eax,eax
  je     <guess_your_age+0x37>
  mov    eax,0xf
  ...
```

Control Flow Graph Recovery

- But this is just about reversible! (undecidable in general though of course!)

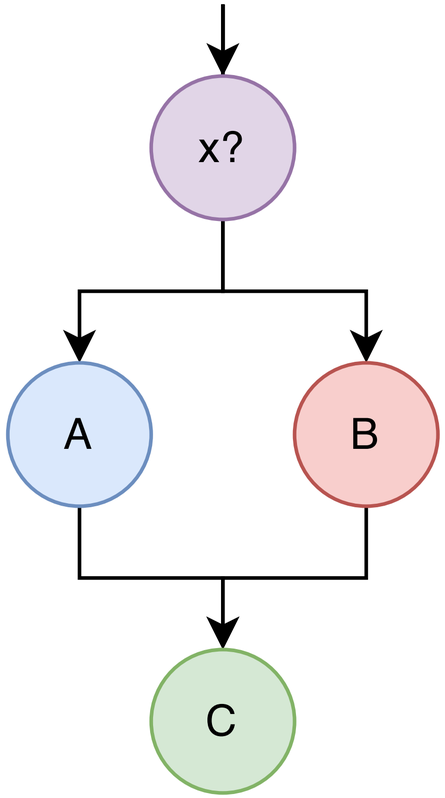
```
guess_your_age:
  push   rbp
  mov    rbp,rsp
  mov    eax,DWORD PTR [rip+0x0]
  test   eax,eax
  je     <guess_your_age+0x26>
  mov    eax,DWORD PTR [rip+0x0]
  test   eax,eax
  je     <guess_your_age+0x1f>
  mov    eax,0x28
  jmp    <guess_your_age+0x4d>
  mov    eax,0x19
  jmp    <guess_your_age+0x4d>
  mov    eax,DWORD PTR [rip+0x0]
  test   eax,eax
  je     <guess_your_age+0x37>
  mov    eax,0xf
  ...
```

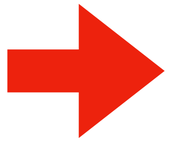




Finding Patterns: if/else

- By looking closely we can usually find **patterns,** which we can map to code.

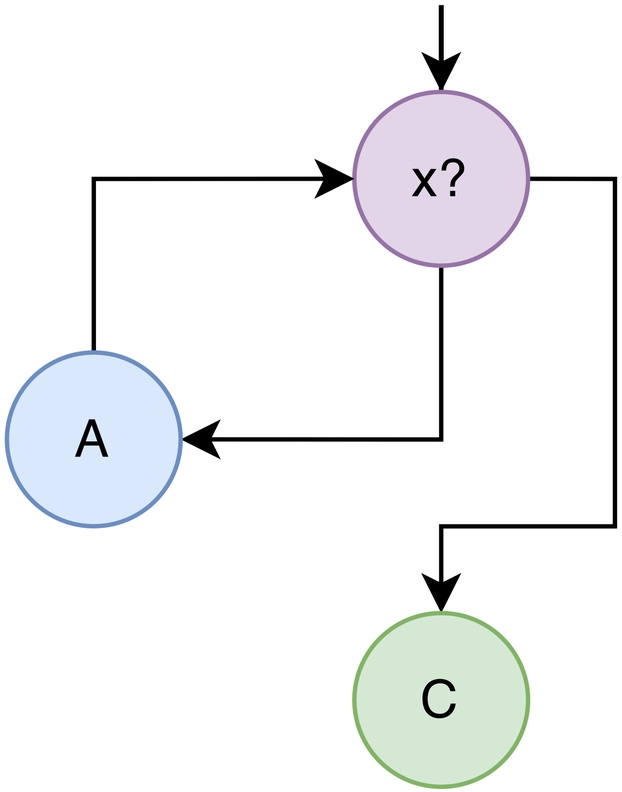


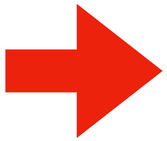


```
if (x)
  A
else
  B
C
```

Finding Patterns: while

- By looking closely we can usually find **patterns,** which we can map to code.

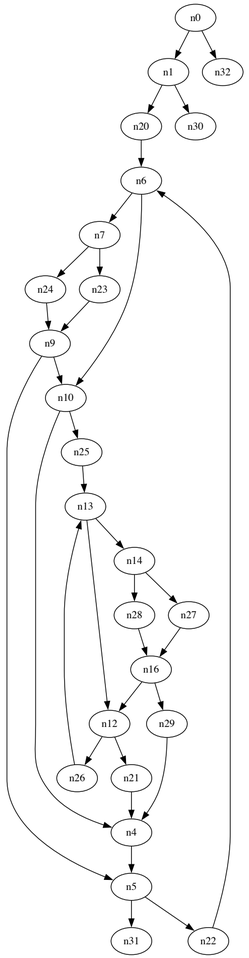


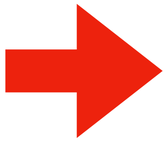


```
while (x)
  A
C
```

Not so simple in practice!

As with most of decompilation, it's an open problem!





```
if (a) return
if (b) return
do {
  if (d) {
    if (e) // ...
    else // ...
    if (f) continue
  }
  // ... etc ...
} while (c)
```

Thanks for listening!

- Hopefully you learned something!
  - What is a compiler?
  - What is a decompiler?
  - How do a decompilers work?
  - Code can be structured as graphs
- And maybe have some questions!
  - Is it ethical to work on decompilers? I personally think so!
  - Does AI trivialise all this?